Benutzerleitfaden für das Programm

AutoTest

Johann Letzel
johann.letzel@johann-letzel.de

17. Januar 2002

Inhaltsverzeichnis

1 Einleitung

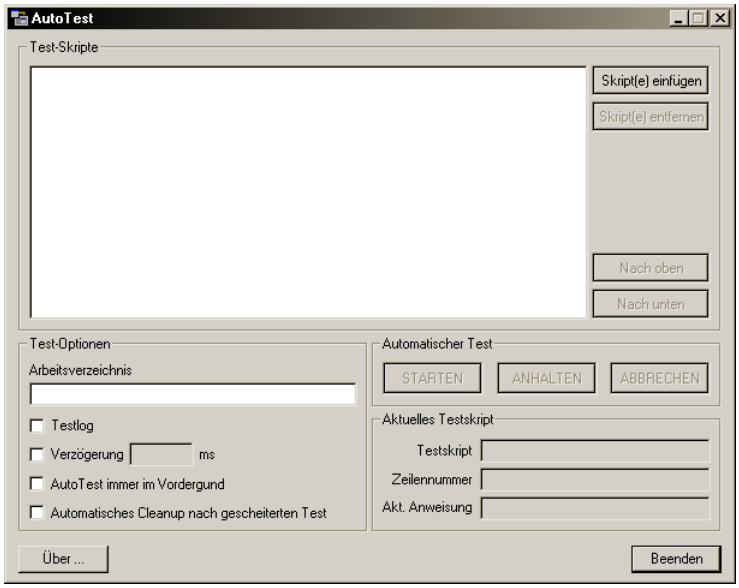
Die Entwicklung von Software beinhaltet nicht nur das Schreiben von Programmen. Sie ist eine Vereinigung verschiedener Konzepte und Tätigkeiten. Darunter fällt auch das Testen von Programmen. Ohne das Testen wäre es schwierig sicher zu stellen, ob die entwickelte Software dem entspricht, was man von ihr erwartet. Eine Spezialform des Testens ist der sogenannte *Regressionstest*. Dieser stellt sicher, daß sich während der Weiterentwicklung bestehender Software die bereits vorhandene Funktionalität erhalten bleibt. Dazu benutzt man eine grobe Auswahl von Testszenarien mit denen möglichst alle Anwendungsszenarien abgedeckt werden. Während der Weiterentwicklung ist es wichtig den Regressionstest nach jeder Änderung des Systems durchzuführen. Allerdings wäre es viel zu zeitintensiv dies per Hand zu erledigen.

Unter diesem Aspekt liegt es nahe diesen Vorgang zu automatisieren und somit den Zeitaufwand für das Testen zu minimieren. Diese Automation ist die Aufgabe des Programms *AutoTest*. Es ist in der Lage mittels sog. Testskripte, die die auszuführenden Aktionen beschreiben, eine Software genauso zu steuern als wenn ein Benutzer mit dieser interagieren würde. Es ist spezialisiert auf die Steuerung von Software mit grafischer Oberfläche der *Windows*^{TM} Betriebssysteme. Darüber hinaus besitzt es die Fähigkeit zusätzliche testbegleitende Programme auszuführen um z.B. Testergebnisse zu analysieren.

Somit ist es einem Entwickler möglich nach Änderungen der Software und/oder der -komponenten schnell und einfach zu überprüfen ob die geforderte Funktionalität noch erhalten ist. Allerdings ist für das Schreiben der oben genannten Testskripte eine gewisses Maß an Erfahrung in Programmierung mittels der *Win32 API* erforderlich.

2 Benutzungsoberfläche (Allgemeine Hinweise)

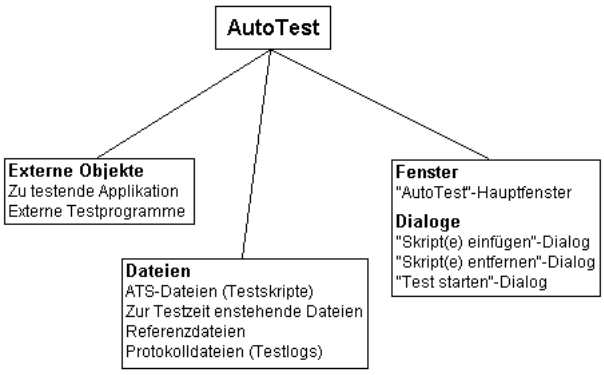
Die Bedienung und die grafischen Elemente des Programms folgen dem Standard der *Windows*^{TM} Betriebssysteme.



Im Folgenden werden die einzelnen grafischen Elemente erläutert.

Element	Bezeichnung	Funktion
123	Eingabefeld	Eingabe von Daten
AutoTest immer im Vordergund	Checkbox	De-/Aktivierung von Einstellungen
Skript(e) einfügen Beenden	Button	Auslösen von Aktionen
Am Ende	Radiobutton	Auswahl von Alternativen

3 Produktstruktur



3.1 Fenster

"AutoTest"-Hauptfenser Erscheint nach dem Programmstart

3.2 Dialoge

"Skript(e) einfügen"-Dialog Erscheint, wenn sich bereits Skripte in der Liste befinden und auf Button "Skript(e) einfügen" geklickt wird

"Skript(e) entfernen"-Dialog Erscheint, wenn auf Button "Skript(e) entfernen" geklickt wird

"Test starten"-Dialog Erscheint, wenn ein Skript in der Liste selektiert ist und auf Button "STARTEN" geklickt wird

3.3 Dateien

ATS-Dateien Sind die eigentlichen Testskripte, die beschreiben welche Inter-/Aktionen mit dem zu testenden Programm für den entsprechenden Test durchgeführt werden sollen

Zur Testzeit entstehende Dateien Sie werden zur Zeit der Testdurchführung vom zu testenden Programm erzeugt

Referenzdateien Solldateien, die mit den zur Testzeit entstehenden Dateien verglichen werden um die korrekte Funktionalität sicher zu stellen

Protokolldateien Werden von *AutoTest* oder von externen Testprogrammen erzeugt, die die Testdurchführung protokollieren

4 Trainingsteil

Durchführung eines typischen Testvorgangs.

4.1 Testskripte für den Testvorgang wählen

- Programm "AutoTest" starten
- Auf Button "Skript(e) einfügen" klicken
- Datei-Dialog "Skript(e) einfügen" öffnet sich
- Testskripte auswählen (auch Mehrfach-Auswahl möglich) und auf Button "Öffnen" klicken
- Ausgewählte Skripte tauchen in der Liste auf

4.2 Reihenfolge der Testskripte ändern

- Skript in Liste duch Klicken selektieren
- Skript durch Klicken auf Button "Nach oben" oder "Nach unten" in der Liste bewegen

4.3 Zusätzliche Testskripte einfügen

4.3.1 Kein Testskript selektiert

- Auf Button "Skript(e) einfügen" klicken
- Datei-Dialog "Skript(e) einfügen" erscheint
- Testskripte auswählen (auch Mehrfach-Auswahl möglich) und auf Button "Öffnen" klicken
- "Skript(e) einfügen"-Dialog erscheint
- Wählen, wo die zusätzlichen Testsskripte eingefügt werden sollen (Anfang oder Ende der Liste)
- Klicken auf Button "OK"
- Ausgewählte Skripte tauchen in der Liste auf

4.3.2 Testskript selektiert

- Auf Button "Skript(e) einfügen" klicken
- Datei-Dialog "Skript(e) einfügen" erscheint
- Testskripte auswählen (auch Mehrfach-Auswahl möglich) und auf Button "Öffnen" klicken
- "Skript(e) einfügen"-Dialog erscheint
- Wählen, wo die zusätzlichen Testsskripte eingefügt werden sollen (Anfang oder Ende der Liste, vor oder hinter dem selektierten Skript)

- Klicken auf Button "OK"
- Ausgewählte Skripte tauchen in der Liste auf

4.4 Testskripte aus der Liste entfernen

- Auf Button "Skript(e) entfernen" klicken
- Dialog "Skript(e) entfernen" erscheint
- Testskripte auswählen (auch Mehrfach-Auswahl möglich) und auf Button "OK" klicken
- Ausgewählte Skripte werden aus der Liste entfernt

4.5 Testeinstellungen vornhemen

- Im Eingabefeld "Arbeitsverzeichnis" den Verzeichnispfad eintragen, wo sich das zu testende Programm befindet
- Die gewünschten Optionen mittels Klicken auf die entsprechenden Checkboxen de-/aktivieren

4.6 Testvorgang starten

4.6.1 Kein Testskript selektiert

- Auf den Button "STARTEN" klicken
- Der Testvorgang beginnt

4.6.2 Testskript selektiert

- Auf den Button "STARTEN" klicken
- Dialog "Testlauf starten" erscheint
- Wählen ab wo der Testvorgang beginnen soll (Am Anfang, ab oder hinter markiertem Testskript)
- Der Testvorgang beginnt

4.7 Testvorgang pausieren und fortsetzen

- Bei laufendem Testvorgan auf Button "ANHALTEN" klicken
- Testvorgang pausiert
- Auf Button "FORTSETZEN" klicken
- Testvorgang wird fortgesetzt

4.8 Testvorgang abbrechen

- Auf Button "ABBRECHEN" oder "Beenden" klicken oder Programm beenden

5 Überblick über die Funktionalität

Die Hauptfunktionalität ist das Steuern von Programmen nach einer ausgewählbaren Menge von Testskripten. Justierbar ist ebenfalls die Reichenfolge der Testskripte. Ein Testskript darf auch mehrmals in einem Testvorgang auftreten. Der Testvorgang läßt sich abbrechen und an gewünschter Stelle fortsetzten. Zusätzlich ist es möglich den Testvorgang anzuhalten.

Sofern die Testskripte die Möglichkeit bieten, wird die zu testende Software nach einem Testvorgang wieder in den Urzustand versetzt um jedem Testszenario die gleichen Voraussetzungen zu bieten. Das sog. *Cleanup* kann sowohl automatisch als auch manuell erfolgen. So besteht die Möglichkeit das zu testende Programm nach einem Fehler im Testvorgang genau zu untersuchen.

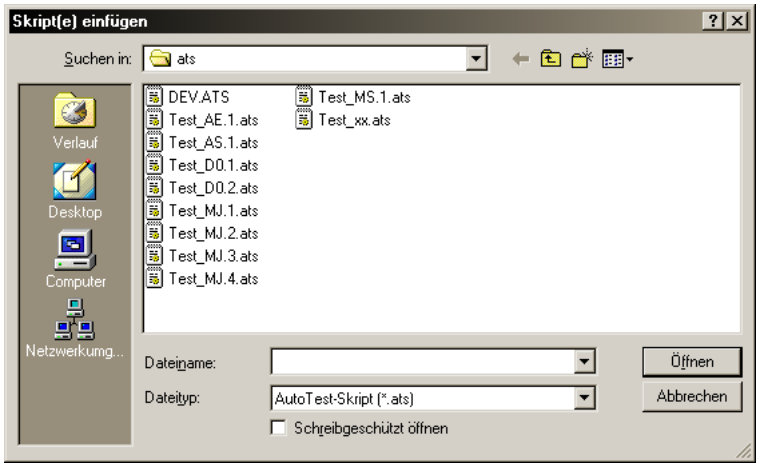
Der gesamte Testvorgang kann auf Wunsch protokolliert werden um evtl. Fehler zu analysieren.

6 Benutzungsoberfläche und Funktionalität

6.1 Zusammenstellen der Testskripte für einen Testvorgang

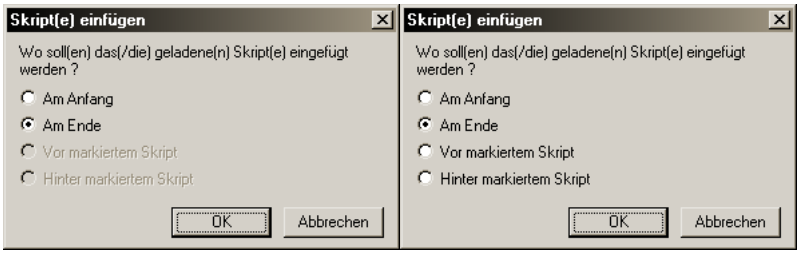
6.1.1 Einfügen von Testskripten in den Testvorgang

Für einen Testvorgang ist festzulegen, welche(s) Testskript(e) auszuführen ist(/sind). Dazu klickt man im Hauptfenster auf den Button "Skript(e) einfügen". Dann öffnet sich ein Dialog "Skripte(e) einfügen", wo die Möglichkeit besteht Dateien auszuwählen. Das Aussehen dieses Dialoges ist abhängig von der Version des *Windows*^{TM} Betriebssystems, das verwendet wird.



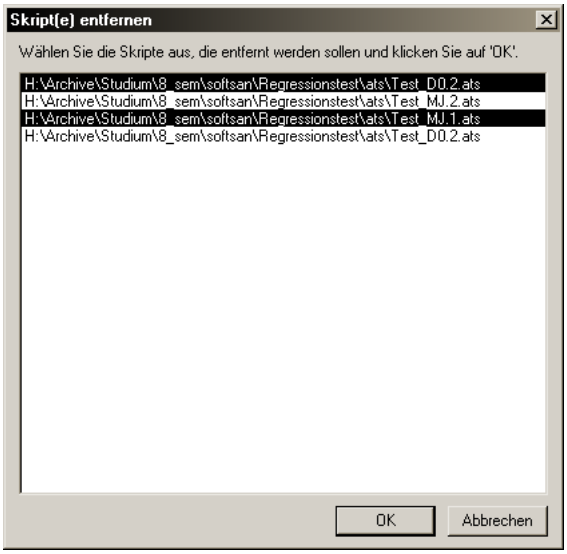
Es ist auch möglich mehrere Testskripte auszuwählen. Klicken mit gedrückter *STRG*-Taste bewirkt das Hinzufügen oder das Entfernen eines Testskriptes zu/von der Auswahl. Klicken mit gedrückter *Umschalt*-Taste ermöglicht die Auswahl zusammanhängender Bereiche von Testskripten. Ein Klicken auf "Öffnen" bewirkt die Übernahmen der Auswahl in die Liste des Testvorgangs.

Falls Skripte eingefügt werden und sich bereits welche in der Liste befinden, erscheint nach der Übernahme der Auswahl ein weiterer Dialog "Skript(e) einfügen", wo die Möglichkeit besteht zu wählen wo die neuen Testskripte eingefügt werden sollen. Es stehen immer die Alternativen "Am Anfang" und "Am Ende" zur Verfügung. Falls in der Liste des Testvorganges ein Skript selektiert ist, gibt es noch zusätzlich die Alternativen "Vor markiertem Skript" und "Hinter markiertem Skript". Ein Testskript kann auch durchaus mehrmals in der Liste auftauchen.



6.1.2 Entfernen von Testskripten aus dem Testvorgang

Um Skripte aus dem Testvorgang zu entfernen klickt man auf den Button "Skript(e) entfernen". Es erscheint ein Dialog "Skripte entfernen".

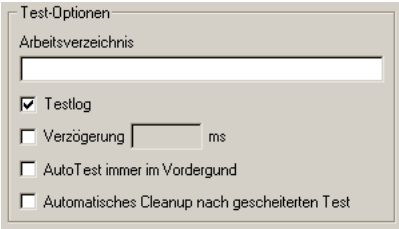


Das Auswählen der zu entfernenden Skripte erfolgt durch Klicken. Auch eine Mehrfachauswahl ist möglich. Durch das Klicken auf den Button "OK" wird die Auswahl übernommen und die entsprechenden Testskripte aus der Liste des Testvorgangs entfernt.

6.1.3 Ändern der Reihenfolge der Testskripte im Testvorgang

Es besteht die Möglichkeit die Reihenfolge der Ausführung der Skripte für den Testvorgang zu bestimmen. Dazu klickt man auf einen Listeneintrag um ihn zu selektieren. Dann klickt man, wenn es erlaubt ist, auf den "Nach oben" oder "Nach unten" um den Eintrag entsprechend um einen Platz zu verschieben.

6.2 Einstellungen für den Testvorgang vornhemen



6.2.1 Arbeitsverzeichnis

Im Eingabefeld "Arbeitsverzeichnis" muß der Verzeichnispfad der zu testenden Software angegeben werden. Dazu klickt man in das Eingabefeld und gibt den Verzeichnispfad ein.

6.2.2 Testlog

Wenn die Checkbox "Testlog" angekreuzt ist, wird während des Testvorgangs zu jedem Testskript ein Testprotokoll angelegt. Die Protokolldateien werden in dem Verzeichnis "log" innerhalb des Test-Verzeichnisbaumes angelegt und erhalten den Namen

"`<Name des Testskriptes>_<Jahr><Monat><Tag>_<Uhrzeit>.log`".

Um es zu deaktivieren klickt man auf die Checkbox. Dann erscheint eine Warnung, daß es nicht sinnvoll ist das Testprotokoll zu deaktivieren. Um es wieder zu aktivieren genügt wieder ein Klick auf die Checkbox.

6.2.3 Verzögerung

Wenn diese Checkbox aktiviert ist, so kann man in dem dahinterliegenden Eingabefeld einen Wert in Millisekunden eingeben, der nach jeder Zeile in einem Testskript gewartet werden soll. Das De-/Aktivieren geschieht durch Klicken auf die Checkbox.

6.2.4 AutoTest immer im Vordergrund

Das wird mit der Checkbox "AutoTest immer im Vordergund" de-/aktiviert. Es bewirkt, daß das Hauptfenster "AutoTest" sich während des Testvorgangs immer im Vordergrund befindet und nicht von normalen Fenstern verdeckt wird. So ist es möglich den Testablauf leichter zu beobachten. Das De-/Aktivieren geschieht mit einfachen Klicks auf die Checkbox.

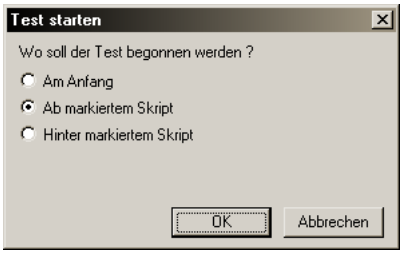
6.2.5 Automatisches Cleanup nach gescheitertem Test

Ist die Checkbox "Automatisches Cleanup nach gescheitertem Test" aktiviert, so wird, wenn ein Testskript scheitert, der Versuch unternommen die zu testende Software wieder in den Originalzustand zu überführen. Wenn die Checkbox hingegen deaktiviert ist, so verbleibt die Software in dem Zustand zum Zeitpunkt des Fehlers. Dann kann es genau untersucht werden und später durch manuelles Aktivieren des Cleanups wieder hergestellt werden.

6.3 Durchführen eines Testvorgangs

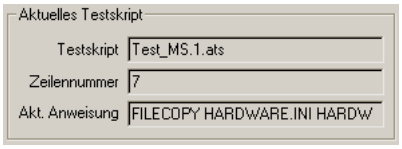
6.3.1 Starten des Testvorgangs

Wenn Testskripte in der Liste für den Testvorgang vorhanden sind, so führt das Klicken auf den Button "STARTEN" zum Start des Testvorgangs. Wenn ein Testskript in der Liste selektiert ist, so erscheint der Dialog "Test starten".



Hier kann bestimmt werden ab wo der Test beginnen soll. Zur Auswahl stehen "Am Anfang", "Ab markiertem Skript" und ggf. "Hinter markiertem Skript". Durch das Klicken auf den Button "OK" wird die Einstellung übernommen und der Testvorgang beginnt. Wenn kein Testskript in der Liste markiert ist, so beginnt der Testvorgang unmittelbat nach dem Klicken auf "STARTEN" am Anfang der Liste.

Während des Testvorganges können die Liste der Testskripte und die Testoptionen nicht verändert werden. Der aktuelle Zustand des Testvorgangs wird in den Feldern "Testskript", "Zeilenumer" und "Akt. Anweisung" angezeigt.



6.3.2 Anhalten und Reaktivieren des Testvorgangs

Soll der Testvorgang für einige Zeit angehalten werden, so genügt ein Klick auf den Button "ANHALTEN". Nun pausiert der Testvorgang, bis man ihn entweder durch das Klicken auf den Button "FORTFAHREN" reaktiviert oder ihn abbricht.

6.3.3 Fehler während des Testvorgangs

Zur Zeit des Testvorganges können drei Arten von Fehlern auftreten: Fehler in der Syntax des Skriptes, Fehler beim Ausführen eines Kommandos innheralb des Skriptes und interne Fehler.

Die erste Art von Fehler tritt auf, wenn das Testskript an sich fehlerhaft ist. Die Ursache hierfür kann ein falsch geschriebenes oder fehlendes Schlüsselwort, falsche Parameter oder ein Fehler in der Struktur sein. Wenn ein solcher Fehler auftritt, ist das entsprechende Skript zu überprüfen.

Die Ursache für die zweite Art von Fehler liegt evtl. in einem nicht vorhandenen Fenster oder einer nicht eingetroffenen Annahme. Wenn ein solcher Fehler

auftritt, kann das auf einen Fehler in der zu testenden Software hindeuten. Sofern der Ablauf des Testskriptes korrekt ist. Wenn ein solcher Fehler auftritt, ist sinnvoll den Testvorgang mit dem entsprechenden Testskript zu wiederholen und ggf. den Fehler in der zu testenden Software zu beheben.

Die internen Fehler treten auf, wenn z.B. nicht in die Protokolldatei geschrieben werden konnte. Das kann mehrere Ursachen haben z.B. zu wenig Speicherplatz oder ein Fehler des Betriebssystems.

Wenn ein Fehler während eines Testvorganges auftritt, erscheinen Meldungen, die über die Art, den Ort und die Ursache des Fehlers informieren. Außerdem wird das Skript, in dem der Fehler auftrat in der Liste selektiert.

Wenn die Option "Automatisches Cleanup nach gescheitertem Test" aktiviert ist, wird die zu testende Software automatisch in den Originalzustand überführt, sofern das Testskript ein Cleanup vorsieht. Falls die Option nicht aktiviert ist, wird der Testvorgang nach dem Auftreten des Fehlers angehalten. Die zu testende Software befindet sich dann im Zustand zum Zeitpunkt des Fehlers und kann untersucht werden. Um anschlißend den Originalzustand zu wieder herzustellen, genügt ein Klick auf den Button "CLEANUP".

Wenn das Testskript ein Cleanup nicht vorsieht, wird darauf hingewiesen und die zu testende Software muß ggf. per Hand in den Originalzustand gebracht werden.

6.3.4 Abbrechen des Testvorgangs

Es ist auch möglich den gesamten Testvorgang abzubrechen. Dazu genügt ein Klick auf den Button "Abbrechen" oder "Beenden". Das Schliessen des Hauptfensters führt ebenfalls zum Abbruch des Testvorganges. Wird versucht den Testvorgang abzubrechen, ercheint eine Warnung, die auch ggf. darauf hinweist, daß das Testskript kein Cleanup vorsieht. Wird ein Cleanup vorgesehen, versucht *AutoTest* den Originalzustand der zu testenden Software automatisch wieder herzustellen.

Wenn versucht wird den Testvorgang abzubrechen, nach dem ein Fehler aufgetreten ist und der Vorgang etweder pausiert oder das automatische Cleanup läuft, erscheint ebenfalls eine Warnung. Wenn der Abbruch bestätigt wird, führt das zu einem sofortigem Abbruch ohne daß ein automatisches Cleanup ausgeführt wird. Das kann zur Folge haben, daß sich dann die zu testende Software NICHT mehr im selben Zustand, wie vor dem Test befindet.

6.3.5 Abschließen des Testvorgangs

Wenn alle Testskripte ohne Fehler ausgeführt worden sind, dann erscheint eine abschließende Meldung und der Testvorgang wird beendet. Nun können wieder Veränderungen an der Testskript-Liste und an den Testoptionen vorgenommen werden.

7 Entwickeln von Testskripten

7.1 Vorwort

Um Tests zu automatisieren ist es nötig die Aktionen, die ein Tester ausführen würde, zu beschreiben. Dazu wurde eine den Anforderungen entsprechende Skriptsprache entwickelt. Sie bietet Funktionen zur Steuerung einer Anwendung, Kopieren von Dateien, Starten von Zusatzprogrammen, usw.

Allerdings sind einige Voraussetzungen nötig um die Skriptsprache einsetzen zu können. Zum einen sind umfangreiche Kenntnisse zur Programmierung der *Win32 API* erforderlich, besonders in Hinblick auf die Steuerung anderer Programme. Außerdem sind ggf. Zugang zu den Quellen und Ressourcen des zu testenden Systems nötig. Es können nur Testvorgänge von Software ausgeführt werden, die unter den *Windows*^{TM}-Betriebssystemen lauffähig sind.

Wenn diese Voraussetzungen erfüllt sind genügt ein einfacher Text-Editor zur Erstellung der Testskripte.

7.2 Struktur der Testskripte

Die Testskripte sind eine Sequenz von Zeilen in denen je ein Kommando steht. Ein Kommando besteht immer aus einem Schlüsselwort und ggf. Paramtern, die sich auf dieses beziehen. Die einzelnen Teile werden durch Kommata getrennt. Hinter dem letzten Paramter bzw. dem Schlüsselwort darf kein Komma stehen. Also in der Form:

```
<Kommando>,<Paramter 1>,<Parameter 2> ...
```

Numerische Parameter können einen Punkt und ein Vorzeichen beeinhalten. Manche Parameter werden als Zeichenketten übergeben und werden mit `"` umschlossen. Innerhalb dieser Zeichenketten gelten spezielle Regeln. Einige Zeichen wie z.B. `\` bzw. `"` werden innerhalb der Zeichenketten durch `\\` bzw. `\"` dargestellt. Die Darstellung lehnt sich an die Programmierpsrache C an und ermöglich auch das Einbringen von Escape-Sequenzen um z.B. einen Zeilenumbruch einzufügen. Es folgt eine Aufstellung der Sequenzen und ihrer Bedeutung:

Sequenz	**Bedeutung**
`\\`	`\` (Backslash)
`\"`	`"` (Anführungszeichen)
`\'`	`'` (Hochkomma)
`\?`	`?` (Fragezeichen)
`\a`	Alert
`\b`	Backspace
`\f`	Form feed
`\n`	New line
`\r`	Carriage return
`\t`	Horizontal tab
`\v`	Vertical tab

Kommentare beginnen immer mit `#` und enden mit dem Ende der Zeile, sofern sich das Zeichen `#` nicht innerhalb einer Zeichenkette befindet.

Die Dateinamen der Skripte müssen immer die Endung *ATS (AutoTest Script)* enthalten, z.B. *TEST_01.ATS* .

7.3 Kommandos der Skriptsprache

Im Folgenden werden die Kommandos, ihre Paramter und ihre Funktion erläutert.

7.3.1 BREAK

Aufruf: `BREAK`

Parameter: keine

Beschreibung: Dieses Kommando dient dazu, eine `LOOP`-Schleife sofort zu beenden. Die Schleife mu allerdings vom Typ `CYCLES` sein. Bei Ausfhrung dieses Kommandos wird direkt hinter das zugehörige `ENDLOOP`-Kommando gesprungen.

Beispiel:

```
LOOP,5,CYCLES,100

 ...

 BREAK  # An dieser Stelle wird die Schleife sofort
        # verlassen

 ...

ENDLOOP
  # An dieser Stelle wird das Skript fortgesetzt
```

7.3.2 CLEANUP

Aufruf: `CLEANUP`

Parameter: keine

Beschreibung: Alle folgenden Kommandos bis zum Ende der Datei sollten zur Wiederherstellung desr zu testenden Software dienen. z.B. Lschen von temporren Dateien, Wiederherstellen wichtiger Dateien. Alle Fehler, die dabei auftreten, werden ignoriert. Falls whrend des Testvorgangs ein Fehler auftritt, wird direkt zu diesem Kommando gesprungen, falls es vorhanden ist und die zuletzt mit `LAUNCH` gstartete Anwendung geschlossen. Alle Testskripte, die die zu testende Software verndern sollten hier einen Teil zur Wiederherstellung beinhalten.

7.3.3 COMPARE

Aufruf: `COMPARE,<ISTWERT>,<SOLLWERT>[,<OPERATOR>]`

Parameter:

`<ISTWERT>`: Reprsentiert einen aktuellen Wert, der mit einem Sollwert verglichen wird. Es knnen folgende Bezeichner verwendet werden: `RESULT`, `WPARAM` und `LPARAM`. Diese Bezeichner stehen fr Ergebnisse von speziellen Kommandos, z.B. `SEND`, `LAUNCHEXTERN`. Nur mit `WPARAM` und `LPARAM` knnen textuelle Vergleiche durchgefhrt werden. Wenn `RESULT` angeben wurde, wird **immer** numerisch verglichen !

`<SOLLWERT>`: Reprsentiert den Sollwert mit dem der Istwert verglichen wird. Hier gibt es mehrere Mglichkeiten. Zum einen kann man einen konstanten Wert z.B. `123` oder `"123"` angeben. Bei einem Wert ohne `"` wird ein numerischer Vegleich zwischen Ist- und Sollwert vorgenommen. Mit `"` hingegen ein textueller.

Aber auch die Variablen `M1`, `M2` und `M3` lassen sich als Sollwerte angeben. Hier wird die Art des Vergleichs vom Typ der Variable bestimmt. Wenn also die Variable z.B. einen numerisches Wert gepseichert hat, wird ein numerischer Vergleich vorgenommen. Es ei denn es wird mit `RESULT` als Istwert verglichen, dann wird immer ein numerischer Vergleich durchgefhrt auch wenn die Variable ein String ist. Den Typ der Variable kann man mit dem Befehl `SAVE` bestimmen.

`<OPERATOR>`: Dieser Parameter ist optional und wird nur bei numerischen Vergleichen beachtet. Wird er nicht angeben, so wird bei numerischen Vergleichen auf Gleichheit geprft. Falls aber nicht auf Gleicheit geprft werden soll, stehen folgende Operatoren zur Auswahl:

`GRT`: Istwert ist grer als Sollwert.

`GEQ`: Istwert ist grer-gleich Sollwert.

`TOL,<TOLERANZ>`: Istwert ist gleich Sollwert mit Toleranz, wobei `<TOLERANZ>` eine positive Konstante ist. D.h. es wird geprft ob folgendes gilt

`<SOLLWERT>-<TOLERANZ>` $\leq$ `<ISTWERT>` $\leq$ `<SOLLWERT>+<TOLERANZ>`

`LEQ`: Istwert ist kleiner-gleich Sollwert.

`LSS`: Istwert ist kleiner als Sollwert.

`NEQ`: Istwert ist ungleich Sollwert.

Beschreibung: Dient zum Vergleich zweier Werte, z.B. von Ergebnissen mit Sollwerten. Wenn der Vergleich scheitert, wird der Testvorgang abgebrochen, auer wenn das `COMPARE`-Kommando die Bedingung fr ein `IF`- oder `IFNOT`-Kommando ist. Dann wird das Ergebnis als "Wahr" oder "Falsch" ausgewertet und fr die Verzweigung benutzt.

Beispiele:

```
COMPARE,RESULT,10       # Ueberpruefen ob 'RESULT' gleich 10
COMPARE,RESULT,12,LSS   # Ueberpruefen ob 'RESULT' kleiner 12
COMPARE,RESULT,M1,TOL,2 # Ueberpruefen ob 'RESULT' innerhalb
                        # der Tolanzgrenzen '2' um den Wert
                        # von 'M1' liegt
COMPARE,WPARAM,"Hallo"  # Ueberpruefen ob 'WPARAM' dem String
                        # 'Hallo' entspricht
```

7.3.4 ELSE

Aufruf: `ELSE`

Parameter: keine

Beschreibung: Alle folgenden Kommandos bis zum zugehörigen `ENDLOOP` des aktuellen `IF`- bzw. `IFNOT`-Blockes dienen als Alternativzweig. Falls die Bedingung zu "Falsch" bzw. "Wahr" auswertet, dann werden alle Kommandos ab dem nchsten `ELSE` (falls vorhanden) innerhalb des Verzweigungsblockes ausgefhrt. Ein `ELSE`-Kommando innerhalb eines Verzweigungsblockes ist optional.

Beispiel: siehe Kommando `IF`

7.3.5 ENDIF

Aufruf: `ENDIF`

Parameter: keine

Beschreibung: Schliet einen `IF`- oder `IFNOT`-Verzweigungsblock ab.

Beispiel: siehe Kommando `IF`

7.3.6 ENDLOOP

Aufruf: `ENDLOOP`

Parameter: keine

Beschreibung: Bildet das Ende eines `LOOP`-Schleifenblockes vom Typ `CYCLES`.

Beispiel: siehe Kommando `LOOP`

7.3.7 FILECOPY

Aufruf: `FILECOPY,<QUELLE>,<ZIEL>[,FORCE]`

Parameter:

`<QUELLE>`**:** Gibt den Pfad der Quelldatei relativ zum Arbeitsverzeichnis an. Es knnen auch Unterverzeichnisse angegeben werden. Zu beachten ist allerdings, da fr ein \ die Zeichenfolge \\ zu schreiben ist. Eine gltige Quelldatei-Name wre z.B. `"REF\\TEST01.REF"`. Der Name mu immer mit " eingeschlossen sein.

`<ZIEL>`**:** Hier gilt das selbe, wie fr den Parameter `<QUELLE>`.

`FORCE`**:** Diese Direktive ist optional. Die ist notwendig, wenn die Zieldatei bereits exisitert und entweder versteckt oder schreibgeschtzt ist. Ohne diese Direktive scheitert das Kopieren in diesem Falle und verursacht einen Fehler. Die Direktive mu mit Vorsicht benutzt werden, da es somit mglich ist Dateien zu manipulieren, die evtl. nicht manipuliert werden sollen!

Beschreibung: Kopiert eine Datei innerhalb dss Arbeitsverzeichnisses.

Beispiele:

```
FILECOPY,"TEST","TEST.BAK"  # Kopiert die Datei 'TEST'
                            # in die Datei 'TEST.BAK'

FILECOPY,"TEST","READONLY",FORCE  # Kopiert die Datei
                                  # 'TEST' in die
                                  # Datei 'READONLY'
                                  # auch wenn letztere
                                  # schreibgeschuetzt
                                  # ist
```

7.3.8 FILEDELETE

Aufruf: `FILEDELETE,<DATEINAME>[,FORCE]`

Parameter:

`<DATEINAME>`**:** Gibt den Pfad der zu lschenden Datei relativ zum Arbeitsverzeichnis an. Es knnen auch Unterverzeichnisse angegeben werden. Zu beachten ist allerdings, da fr ein \ die Zeichenfolge \\ zu schreiben ist und da der Name in " eingeschlossen sein mu. Also z.B. `"FILE.BAK"`.

`FORCE`**:** Ist eine optionale Direktive, die angegeben werden sollte, wenn die zu lschende Datei entweder versteckt oder schreibgeschtzt ist. Die Direktive mu mit Vorsicht benutzt werden, da es somit mglich ist Dateien zu lschen, die evtl. nicht gelscht werden sollen!

Beschreibung: Lscht die angegebene Datei im Arbeitsverzeichnis. Falls die Datei nicht existiert, tritt ein Fehler auf.

Beispiele:

```
FILEDELETE,"DEL"  # Loescht die Datei 'DEL'

FILEDELETE,"DEL",FORCE  # Loescht die Datei 'DEL' auch wenn
                        # diese schreibgeschuetzt ist
```

7.3.9 FILEEXISTS

Aufruf: `FILEEXISTS,<DATEINAME>`

Parameter:

`<DATEINAME>`**:** Der Name der Datei, deren Existenz zu berprfen ist. Der Dateiname kann auch Pfadangaben enthalten, wobei ein \ durch \\ zu ersetzen ist. Der Name bezeichnet eine Datei relativ zum Arbeitsverzeichnis. Der Dateiname mu von " eingeschlossen sein. Also z.B. `"RESULT.DAT"`

Beschreibung: Prft ob die angegebene Datei vorhanden ist. Wenn die Datei nicht vorhanden ist, tritt ein Fehler auf, auer das `FILEEXISTS`-Kommando ist eine Bedingung fr ein `IF`- oder `IFNOT`-Kommando. Dann wird es zu "Wahr" oder "Falsch" ausgewertet.

Beispiel:

```
FILEEXISTS,"EXIST"  # Ueberpruefen ob die Datei 'EXIST'
                    # vorhanden ist
```

7.3.10 IF und IFNOT

Aufruf: `IF` oder `IFNOT`

Parameter: keine

Beschreibung: Beginnt einen Verzweigungsblock. Das folgende Kommando wird als Bedingung interpretiert. Wenn die Bedingung fr `IF` zu "Wahr" auswertet, dann werden die Kommandos hinter der Bedingung bis zu einem `ELSE` oder bis zum Ende des Verzweigungsblockes ausgefhrt. Wenn die Bedingung hingegen zu "Falsch" auswertet, dann werden die Kommandos hinter dem `ELSE` oder dem Ende des Verzweigungsblockes ausgefhrt. Bei `IFNOT` verhlt es sich umgekehrt.

Als Bedingung sind die Kommandos `COMPARE` und `FILEEXISTS` zulssig. Es sind maximal 16 Verzweigungsebenen erlaubt.

Beispiele:

```
IF
  # Die Anweisung an dieser Stelle wird, wenn moeglich
  # als Bedingung ausgewertet

  # Die Anweisungsfolge an dieser Stelle wird ausgefuehrt,
  # wenn die Bedingun zu 'Wahr' auswertet
ENDIF

IF
  # Die Anweisung an dieser Stelle wird, wenn moeglich
  # als Bedingung ausgewertet

  # Die Anweisungsfolge an dieser Stelle wird ausgefuehrt,
  # wenn die Bedingun zu 'Wahr' auswertet
ELSE
  # Die Anweisungsfolge an dieser Stelle wird ausgefuehrt,
  # wenn die Bedingun zu 'Falsch' auswertet
ENDIF

IFNOT
  # Die Anweisung an dieser Stelle wird, wenn moeglich
  # als Bedingung ausgewertet

  # Die Anweisungsfolge an dieser Stelle wird ausgefuehrt,
  # wenn die Bedingun zu 'Falsch' auswertet
ELSE
  # Die Anweisungsfolge an dieser Stelle wird ausgefuehrt,
  # wenn die Bedingun zu 'Wahr' auswertet
ENDIF
```

7.3.11 INSERTLOG

Aufruf: `INSERTLOG,<DATEINAME>`

Parameter:

`<DATEINAME>`**:** Gibt den Namen der Datei relativ zum Arbeitsverzeichnis an, die in das Testprotokoll eingefgt werden soll. Der Name kann auch Pfadangaben enthalten. Allerding mu anstatt `\` die Zeichenfolge `\\` benutzt werden und der Name mu von `"` eingeschlossen sein. Also z.B. `"RESULT.LOG"`.

Beschreibung: Fgt eine (Text-)Datei in das Testprotokoll ein. Die angegebene Datei sollte im ASCII-Fomat vorliegen. Eine solche Datei knnte z.B. ein Ergebnisprotokoll eines externen Testprogrammes sein.

Beispiel:

```
INSERTLOG,"EXTLOG"  # Fuegt den Inhalt der Datei 'EXTLOG'
                    # in das aktuelle Testlog ein
```

7.3.12 LAUNCH

Aufruf: `LAUNCH,<AUFRUF>,<FENSTERTITEL>`

Parameter:

`<AUFRUF>`**:** Gibt den Aufruf einer Anwendung relativ zum Arbeitsverzeichnis an. Der Aufruf mu von `"` umschlossen sein und anstatt `\` ist `\\` zu benutzen. Es knnen auch Optionen fr den Start der Anwendung angegeben werden. Z.B. `"PROGRAMM.EXE -i"` . Falls das Programm nicht gestartet werden konnte, tritt ein Fehler auf.

`<FENSTERTITEL>`**:** Beschreibt den Titel des Hauptfensters der zu startenden Anwendung. Der Titel mu von `"` eingeschlossen sein, z.B. `"Fenster"` . Falls das Fenster nach dem Start nicht erscheint, tritt ein Fehler auf.

Beschreibung: Mit diesem Kommando sollte die zu testende Anwendung gstartet werden.

Beispiel:

```
LAUNCH,"TEST.EXE -a -h","Titel"  # Startet das Programm
                                 # 'TEST.EXE' mit den
                                 # Optionen '-a -h' und
                                 # das Fenster mit dem
                                 # Titel 'Titel' wird
                                 # als Haupfenster
                                 # betrachtet
```

7.3.13 LAUNCHEXTERN

Aufruf: `LAUNCHEXTERN,<AUFRUF>,<TIMEOUT>`

Parameter:

`<AUFRUF>`**:** Gibt den Aufruf einer Anwendung relativ zum Arbeitsverzeichnis an. Der Aufruf mu von " umschlossen sein und anstatt \ ist \\ zu benutzen. Es knnen auch Optionen fr den Start der Anwendung angegeben werden. Z.B. `"PROGRAMM.EXE -i"` . Falls das Programm nicht gestartet werden konnte, tritt ein Fehler auf.

`<TIMEOUT>`**:** Beschreibt die Zeitspanne, die auf das Beenden der gestarteten Anwendung gewartet werden soll. Die Zeitspanne wird in Millisekunden angegeben. Mit `FOREVER` wird auf das Beenden gewartet unabhngig von der Dauer. Falls die Anwendung nach Ablauf der Zeitspanne nicht beendet ist, tritt ein Fehler auf.

Beschreibung: Es wird eine externe Anwendung gestartet. Solche Anwednungen knnten externe Testprogramme sein. Der Rckgabewert der Anwednung, wird in der Ergebnisvariablen `RESULT` gespeichert und kann mit `COMPARE` untersucht werden.

Beispiel:

```
LAUNCHEXTERN,"EXT.EXE -i",5000  # Startet das Programm
                                # 'EXT.EXE' mit der
                                # Option '-i' und wartet
                                # maximal 5000 ms auf
                                # dessen Beendigung
```

7.3.14 LOOP

Aufruf: `LOOP,<DAUER>,<TYP>,<VERZGERUNG>`

Parameter:

`<DAUER>`**:** Gibt die Gesamtdauer an, die die Schleife abgearbeitet werden soll. Bei Schleifen vom Typ `CYCLES` ist es die Anzahl der Zyklen, die die Schleife durchlaufen soll. Bei `TIME`-Schleifen hingegen wird die Gesamtdauer in Millisekunden angegeben und beschreibt wie lange die Schleife wiederholt werden soll.

`<TYP>`**:** Es gibt zwei Arten von Schleifen. Zum einen den Typ `CYCLES`. Solche Schleifen werden so oft durchlaufen, wie es im Parameter `<DAUER>` angegeben wurde. Dem `LOOP` drfen dann mehrere Kommandos folgen, die durch ein `ENDLOOP` abgeschlossen werden. Alle Kommandos bis zum `ENDLOOP` werden bei jedem Zyklus wiederholt.

Der andere Typ sind die `TIME`-Schleifen. Hier wird die Schleife solange durchlaufen, wie die in `<DAUER>` angegebene Zeitspanne noch nicht abgelaufen ist. Solchen Schleifen darf nur ein Komando folgen ohne anschlieendes `ENDLOOP`. Das folgende Kommando wird dann bei jedem Durchlauf wiederholt.

<VERZGERUNG>: Beschreibt die Zeit im Millisekunden, die nach jedem Zyklus gewartet werden soll. Diese Wartezeit geht bei TIME-Schleifen in die Berechnung der Gesamtdauer mit ein.

Beschreibung: Bildet den Anfang eines Schleifenlockes. Wenn in Schleifen vom Typ CYCLES ein BREAK ausgefhrt wird, so ist die Schleife sofort beendet und das Ausfhren der Kommandos wird hinter dem nchsten ENDLOOP fortgesetzt. Es sind maxiamal 16 Schleifenebenen zulssig.

Beispiele:

```
LOOP,5,CYCLES,100
   # Fuehrt die hier eingefuegte Anweisungsfolge 5-mal
   # aus und wartet nach jedem Zyklus 100 ms
ENDLOOP

LOOP,5000,TIME,100
   # Fuehrt die hier eingefuegte Anweisung solange aus
   # bis 5000 ms abgelaufen sind und nach jedem Zyklus
   # wird 100 ms gewartet
```

7.3.15 MESSAGE

Aufruf: MESSAGE,<MELDUNG>

Paramater:

<Meldung>: Gibt die Meldung an, die dem Benutzer angezeigt werden soll. Sie mu von " umgeben sein, z.B. "Das ist eine Medlung !".

Beschreibung: Erzeugt eine Meldung in Form einer Messagebox mit einem "OK" Button.

Beispiel:

```
MESSAGE,"Hello World"  # Erzeugt ein Nachrichtenfenster
                       # mit dem Text 'Hello World'
```

7.3.16 POST

Aufruf: POST,<FENSTER>,<CONTROL>,<MESSAGE>,<WPARAM>,<LPARAM>

Parameter: siehe Kommando SEND

Beschreibung: Dieses Kommando schickt eine Nachricht mittels der „Win32 API"-Methode PostMessage() an ein Fenster/Control. Allerdings wird hier kein Rckgabewert in der Ergebnisvariablen RESULT gespeichert. Sonst ist dieses Kommando quivalent zum Kommando SEND.

Beispiele: siehe Kommando SEND

7.3.17 QUESTION

Aufruf: `QUESTION,<FRAGE>`

Parameter:

`<FRAGE>`**:** Die Meldung/Frage, die dem Benutzer angezeigt werden soll. Der Text mu in `"` eingeschlossen sin, z.B. `ind Sie sicher ?"`.

Beschreibung: Zeigt dem Benutzer eine Meldung/Frage in einer Messagebox mit einem "Ja"- und einem "Nein"-Button. In `RESULT` wird das Ergebnis in Abhngigkeit von der Antwort des Benutzers gespeichert. Den Wert 6, falls die Antwort "Ja" ist bzw. den Wert 7 bei "Nein".

Beispiel:

```
QUESTION,"Sind Sie sicher ?"  # Erzeugt ein Fenster,
                              # das die Frage
                              # 'Sind sie sicher ?'
                              # dem Benutzer zeigt
                              # und auf eine Antwort
                              # wartet
```

7.3.18 SAVE

Aufruf: `SAVE,<QUELLE>,<SPEICHER>,<MODUS>`

Parameter:

`<QUELLE>`**:** Gibt den Wert an, der zwischengespeichert werden soll. Zur Auswahl stehen `RESULT`, `WPARAM` und `LPARAM`.

`<SPEICHER>`**:** Beschreibt die Variable, wo der Wert gespeichert werden soll. Zulssige Angaben sind `M1`, `M2` und `M3`.

`<MODUS>`**:** Gibt an, wie der Wert gespeichert werden soll. Es gibt die Modi `NUM` fr numerisches und `STR` fr textuelles Speichern. Die Art der Speicherung kann spter den Vergleich mittels `COMPARE` beeinflussen. Siehe `COMPARE`.

Beschreibung: Speichert den spezifizierten Wert in einer Variablen und nimmt ggf. Konvertierungen zwischen textuellen und numerischen Formaten vor.

Beispiele:

```
SAVE,LPARAM,M1,NUM  # Wandelt den in 'LPARAM' gespeicherten
                    # String in eine Zahl und speichert
                    # diese in 'M1'

SAVE,RESULT,M2,STR  # Wandelt die in 'RESULT' gespeicherte
                    # Zahl in einen String und speichert
                    # diesen in 'M2'
```

7.3.19 SEND

Aufruf: `SEND,<FENSTER>,<CONTROL>,<MESSAGE>,<WPARAM>,<LPARAM>`

Parameter:

`<FENSTER>`**:** Spezifiziert das Zielfenster oder dessen Control an das die Nachricht geschickt werden soll. Wenn `MAIN` angegeben wird, ist das Ziel das Hauptfenster, da durch `LAUNCH` ermittelt wurde. Es kann aber auch ein `"` eingeschlossener Titel angegeben werden. Dann wird das Fenster als Ziel genommen, in dessen Titel die angegebene Zeichenfolge vorkommt. Es ist also darauf zu achten, da der angegebene Titel fr das gesamten System eindeutig ist. Als Beispiel fr das Fenster `"Fenster"` ist es mglich `"Fenster"` oder auch nur `"Fenst"` anzugeben, sofern das eindeutig ist.

`<CONTROL>`**:** Gibt evtl. das Control an, an das die Nachricht geschickt werden soll. Wenn hier `NONE` angegeben wird, geht die Nachricht direkt an das in `<FENSTER>` spezifizierte Fenster. Wenn das Ziel ein Control ist, dann mu hier der numerische Wert stehen, der der ID des Controls entspricht, z.B. `123`.

`<MESSAGE>`**:** Ist der numerische Wert der Nachricht, die geschickt werden soll. Die Nachricht sollte einer der „Win32 API“ entsprechen.

`<WPARAM>` **und** `<LPARAM>`**:** Deren Werte hngen von der in `<MESSAGE>` spezifizierten Nachricht ab und werden von der „Win32 API“ vorgegeben. Bei bestimmten Nachrichten, verlangt die „Win32 API“, da evtl. ein Zeiger auf einen String (`LPSTR`) bergeben werden soll. In einem solchen Fall gengt es dann die Zeichenkette `""` anzugeben. Nach der Ausfhrung von `SEND` befindet sich dann der Ergebnisstring in der Ergebnisvariablen `WPARAM` bzw. `LPARAM`.

Beschreibung: Sendet eine „Win32 API“-Nachricht an ein Fenster/Control. Dazu wird die Methode `SendMessage()` verwendet. Der Rckgabewert dieser Methode wird in der Ergebnisvariablen `RESULT` gespeichert. Manche Nachrichten drfen nicht mittels `SendMessage()` versendet werden. In einem solchen Falle ist das Kommando `POST` zu verwenden.

Beispiele:

```
SEND,MAIN,NONE,273,0,0  # Sendet die Nachricht '273' an
                        # an das Hauptfenster mit den
                        # Parametern '0,0'

SEND,MAIN,1033,16,4,0  # Sendet die Nachricht '16' an das
                       # Control '1033' im Hauptfenster
                       # mit den parametern '4,0'

SEND,"Dialog",678,13,260,"" # Sendet die Nachricht '13'
                            # an das Control '678' im
                            # Fenster mit dem Titel
                            # 'Dialog' mit den Parametern
                            # '260' und einem Puffer fuer
```

```
                                    # eine Zeichenkette, die
                                    # in 'LPARAM' gespeichert wird
```

7.3.20 TESTFILECOPY

Aufruf: `FILECOPY,<QUELLE>,<ZIEL>[,FORCE]`

Parameter:

`<QUELLE>`**:** Gibt den Pfad der Quelldatei relativ zum Testverzeichnisbaum an. Es knnen auch Unterverzeichnisse angegeben werden. Zu beachten ist allerdings, da fr ein \ die Zeichenfolge \\ zu schreiben ist. Eine gltige Quelldatei-Name wre z.B. `"REF\\TEST01.REF"`. Der Name mu immer mit " eingeschlossen sein.

`<ZIEL>`**:** Hier gilt das selbe, wie fr den Parameter `<QUELLE>`. Allerdings bezieht sich der Name hierbei auf eine Datei relativ zum Arbeitsverzeichnis.

`FORCE`**:** Diese Direktive ist optional und ist notwendig, wenn die Zieldatei bereits exisitert und entweder versteckt oder schreibgeschtzt ist. Ohne `FORCE` scheitert das Kopieren in diesem Falle und verursacht einen Fehler. Die Direktive mu mit Vorsicht benutzt werden, da es somit mglich ist Dateien zu manipulieren, die evtl. nicht manipuliert werden sollen!

Beschreibung: Kopiert eine Datei aus den Testverzeichnisbaum in das Arbeitsverzeichnis.

Beispiele:

```
TESTFILECOPY,"TEST.REF","TEST.DAT"  # Kopiert die Datei
                                    # 'TEST.REF' aus dem
                                    # TVZB als 'TEST.DAT'
                                    # in das Arbeits-
                                    # verzeichnis

TESTFILECOPY,"SRC","READ",FORCE  # Kopiert die Datei
                                 # 'SRC' aus dem TVZB
                                 # als 'READ' in das
                                 # Arbeitsverzeichnis
                                 # auch wenn letztere
                                 # existiert und schreib-
                                 # geschuetzt ist
```

7.3.21 WAIT

Aufruf: `WAIT,<TIMEOUT>`

Parameter:

`<TIMEOUT>`**:** Gibt die Zeitdauer in Millisekunden an, die gewartet werden soll.

Beschreibung: Wartet die in `<TIMEOUT>` angegebene Zeit bevor mit der Ausführung des Testskriptes fortgefahren wird.

Beispiel:

```
WAIT,1500  # Wartet 1,5 Sekunden
```

7.3.22 WAITFORWINDOW

Aufruf: `WAITFORWINDOW,<FENSTERTITEL>,<TIMEOUT>`

Parameter:

`<FENSTERTITEL>`**:** Gibt den Titel des Fensteras an auf das gewartet werden soll. Der Titel muß in `"` eingeschlossen sein. Wenn der Titel des Fensters, das erscheinen soll z.B. `"Fenster"` ist, dann kann man entweder `"Fenster"` oder auch nur `"Fenst"` angeben sofern diese Angabe eindeutig ist.

`<TIMEOUT>`**:** Gibt die Zeitspanne in Millisekunden an, die auf das Erscheinen des Fensters maximal gewartet werden soll. Wird `FOREVER` angegeben, wird auf das Erscheinen des Fensters gewartet unabhängig von der Zeitdauer.

Beschreibung: Wartet die in `<TIMEOUT>` angegebene Zeitspanne auf das Fenster mit dem Titel `<FENSTERTITEL>`. Wenn das Fenster nach der Zeitspanne nicht erschienen ist, tritt ein Fehler auf.

Beispiel:

```
WAITFORWINDOW,"Fenster",5000  # Wartet fuer 5 Sekunden auf
                              # das Fenster mit dem Titel
                              # "Fenster"
```

8 Glossar

A

Arbeitsverzeichnis Der Verzeichnispfad, wo sich die zu testende Anwendung befindet und alle Testaktionen ausgeführt erden.

C

Cleanup Beschreibt das "Aufräumen" nach einem (gescheiterten) Test. Alle Testszenarien sollten mittels eines *Cleanups* die zu testende Software wieder in den Zustand bringen, in dem sie vor dem Test war.

K

Klick, klicken Das einmalige Drücken der linken Maustaste, während sich der Mauszeiger über dem gewünschten Objekt befindet.

M

Messagebox Ein Nachrichtenfenster, däs die Aufgabe hat den Benutzer zu informaieren, wenn z.B ein Fehler aufgetreten ist.

T

Testskript Eine Datei die für je ein Testszenario die Aktionen beschreibt, die für den Test notwendig sind.

Test-Verzeichnisbaum, TVZB Die Verzeichnisstruktur, die alles Benötigte für den Test beeinhaltet, z.B. *AutoTest*, Testskripte, Referenzdateien, usw.

Testvorgang Ein vom Tester durcheführter Vorgang, der auch mehrere Testszenarien umfassen kann. Das ist der eigentliche Test.